命令模式

> 有时为了需要向某些对象发送请求, 但是并不知道请求的接收者是谁, 也不知道被请求的操作是什么, 此时希望用一种松耦合的方式来设计程序, 使得请求发送者和请求接收者能够消除彼此之间的耦合关系

传统的命令模式简单的实现

```
function setCommand(receiver){
    receiver.execute()
}

function turnRight(){
    return {
        execute: function(){
            console.log('右拐')
        }
    }
}

function turnLeft(){
    return {
        execute: function(){
            console.log('左拐')
        }
    }
}

setCommand(new turnRight()) // 右拐
setCommand(new turnLeft()) // 左拐
```

这里将一个个命令封装起来, 每个命令都有一个 execute 方法, 要执行命令时直接将对象以参数形式给命令设置方法即可

js 版本的命令模式

命令模式是不清楚未来执行者会操作什么命令而将执行者和接收者分离开来的模式, 使得命令的添加和调用变得更加简单, 在 js 中, 函数回调也有类似的功能

这里借用 `javascript 设计模式与开发实践` 一书中的实例

```
let bindClick = function (btn, fn){
    btn.onclick = fn
}

let menuBar = {
    refresh: function(){}
}

let subMenu = {
    add: function(){},
    del: function(){}
}
```

```
bindClick(btn1, menuBar.reflash)
bindClick(btn2, subMenu.add)
bindClick(btn3, subMenu.del)
```

作用

- 通过引入中间件降低系统的耦合度
- 拓展性好, 增加命令和删除命令十分方便, 采用命令模式增加和删除命令并不会影响其他类, 遵循开放封闭原则
- 方便在现有的功能上拓展其他功能
- 和其他模式组合能够发挥意向不到的效果, 比如和组合模式实现宏命令, 和备忘录模式实现撤销与恢复